공학석사학위논문

머신러닝 모델 유동화와 자동화 거래 시장 설계

Monetization of Machine Learning Model and Architecture for Automated Market

2022년 8월

서울대학교 대학원

산업공학과

조 현 흠

머신러닝 모델 유동화와 자동화 거래 시장 설계

Monetization of Machine Learning Model and Architecture for Automated Market

지도 교수 조 성 준

이 논문을 공학석사 학위논문으로 제출함
2022년 8월

서울대학교 대학원
산업공학과
조 현 흠

조현흠의 공학석사 학위논문을 인준함
2022년 8월

위 원 장 윤명환 (인)

부위원장 조성준 (인)

위 원 이재욱 (인)

초 록

최근 머신러닝 관련 많은 연구와 논문들이 SOTA(State-of-the-Art)급의 성능을 주장하지만 해당 실험환경에서만 최적화된 결과가 나오는 경우도 많고, 다양한 평가 지표가 있으나 해당 지표들만으로는 클라이언트 입장에서 모델의 우열을 평가하는 것이 매우 어렵다. 실제로 논문에서 SOTA 모델이 산업 응용에서는 좋지 못하는 성능을 보이는 경우가 많다. 이에 STO(Security Token Offering), AMM(Automated Market Maker)와 같은 블록체인 기반의 자산 유동화 및 자동화 거래 시장 대한 연구를 진행하였고, 머신러닝 모델 자체를 유동화하여 시장 가격 논리에 맞게 거래되고 평가될 수 있으면서도 모델의 영리적 가치를 보호하는 아키텍처를 제안하였다. 또한 이를 논리적으로 입증하는 프로그램을 구현하였다.

주요어 : 머신러닝, 자동화 거래 시장, 블록체인, 증권형 토큰 공개
학 번 : 2020-24316

목 차

표 목차

그림 목차

제 1 장 서론

1.1. 연구의 배경

머신러닝 모델들이 빠르게 정교화되고 발전함에 따라 산업계에서 실제적으로 적용되고 활용되는 사례가 확산되고 있다. 이에 기업의 머신러닝에 대한 수요는 폭발적으로 증가하고 있지만, 이러한 모델의 효율성과 효과성을 정량적이고 직관적으로 평가하는 지표, 특히 클라이언트 입장에서 어떠한 모델이 가장 우월한지에 대해 평가하는 것은 매우 어렵다. 현재의 머신러닝 모델을 활용하고 거래되는 시장은 다소 비효율적이며 블랙박스와 가까운 모습을 종종 보인다.

이러한 문제점의 원인은 특히 많은 머신러닝 연구원들이 스스로의 모델이 SOTA(State-of-the-Art, 해당 도메인 최고의 성능)을 성취했다고 논문을 통해 주장하지만, Pablo et al [1]에 따르면 실험 환경 및 사용된 데이터 혹은 평가 지표 등이 해당 모델의 성능을 최적화하도록 설계한 경우도 빈번하여, 실제로 산업계에서 응용하여 다른 데이터들을 분석하는 데에 모델을 활용할 경우 성능이 주장하는 것만큼 나오지 않는 경우가 많다. 이러한 것을 분간해내는 것은 쉽지 않고, 특히 머신러닝을 연구하는 입장이 아닌 수요하는 입장인 고객들은 더욱이 분간하는 데에 어려움을 겪을 수 있다.

이처럼 머신러닝 모델을 수요하고 공급하는 시장이 다소 폐쇄적이고 비효율적인 특성을 내포하고 있다는 점은 머신러닝 산업계 전반의 성장에 큰 악영향을 줄 수 있다. 이에 이러한 구조를 탈피하고 보다 효과적이고 공개적으로 머신러닝 모델을 거래할 수 있는 시장에 대해 고민해보게 되었고, 본 연구를 진행하게 되었다.

1.2 문제 정의 및 연구 목적

본 연구에서는 기존의 머신러닝의 논문들이 모델을 최적화하는 데에 초점을 두는 것과 달리 머신러닝 모델들이 보다 공정하고 효과적으로

평가받고 거래될 수 있는 시장 그 자체를 조성하는 것을 목적으로 연구를 진행하였다.

머신러닝 모델 역시 하나의 상품이라고 본다면, 재화가 공정한 가치를 평가받는 최선의 방법 중 하나는 공정한 자유 시장에 의해 평가받는 것이다. 즉, 순수한 머신러닝 모델 자체가 유동화되어 거래될 수 있는 형태가 되고, 이러한 모델이 수요와 공급에 따라 거래되면서 소비자들이 해당 모델로 특정 문제들을 해결하고 해당 모델의 가치를 직접적으로 체감함에 따라 자유 시장 논리에 의거하여 적절한 시장 가격으로 평가받도록 하는 것이다.

또한 스마트 컨트랙트의 함수는 블록체인 상에서 실현되는 반면, 머신 러닝의 학습 과정은 일반적인 컴퓨터 서버에서 진행됨에 따라 두 환경 간의 데이터 전송에 있어 데이터 무결성을 입증해줄 수 있는 신뢰할 수 있는 기술적인 방법이 필요하다.

종합적으로 머신러닝 모델이 가장 직관적이고 효과적으로 평가받는 방법은 자유 시장의 시장 가격 논리에 투명하게 맡기는 것이며, 이러한 시장을 조성하기 위해서는 머신러닝 모델이 거래될 수 있는 방식으로 유동화되고, 머신러닝의 학습 결과를 블록체인 상에서 신뢰할 수 있도록 하는 오라클 서비스를 활용한 뒤, 머신 러닝의 모델이 적절한 가치를 지불하는 본 연구의 목적이다.

1.3 논문 구조 및 기여

본 논문은 전체 5장으로 구성된다. 제 2장에서는 본 연구에 도움을 줄 수 있는 기존의 관련 연구들에 대해 조사한 바를 분석한다. 제 3장에서는 본 논문에서 제안하는 아키텍처에 대한 방법론에 대해 살펴본다. 제 4장에서는 제안한 아키텍처를 실질적으로 코드로 구현하고 실행한 뒤 결과를 분석한다. 제 5장에서는 본 연구를 통해 얻은 결론과 추후의 바람직한 연구 방향에 대해 논의한다. 본 논문이 기여하는 바는 다음과 같다. 첫째, 연구가 진행되지 않았던 머신러닝 모델을 거래하는 효율적 시장에 대한 연구를 진행하였다. 둘째, 해당 시장에 대한 연구를 위해 블록체인과 스마트 컨트랙트 등을 활용하는 학제적 접근을 활용하였다. 셋째, 단순히 아키텍처를 설계하는 것을 넘어서 자동화

코드를 구현하여 실현 가능성을 입증한다.

제 2 장 관련 연구

2.1. 블록체인의 불변성과 합의 매커니즘

블록체인은 암호화폐의 거래를 기록하는 분산 장부로 암호학 기술을 사용하여 탈중앙화된 peer-to-peer의 불변 데이터베이스로서 기능하는 새로운 형태의 자료 구조이자 시스템 아키텍처이다. 데이터베이스 역할을 하는 것과 동시에 암호학적인 합의를 통해 거래를 증명하고 기록하는 역할을 하는 peer를 노드라고 하며, 업데이트 단계마다 모든 노드가 동일한 거래 상태에 대한 합의에 이르렀을 때에 거래 상태들이 블록에 담기게 된다. 따라서 이러한 합의를 하는 매커니즘이 블록체인의 핵심 중 하나라고 할 수 있다.

비트코인으로써 블록체인이라는 개념을 최초로 제안한 Nakamoto[2]는 블록과 블록 사이 발생하는 수천개의 트랜잭션을 2개씩 묶어서 Eastlake and Jones[3]가 제안한 SHA 256 암호화 방식으로 해시 값을 계산하고, 이렇게 나온 해시 2개를 다시 해싱하는 방식으로 최종적으로 하나의 해시 값인 루트 해시가 나올 때까지 연산을 반복하는 머클 트리의 자료 구조를 사용함한다. 이를 통해 트랜잭션 하나의 아주 세부적인 내용만이 달라져도 루트 해시 값이 아예 달라지도록 하여 검증에 최적화되도록 하였다.

또한 작업 증명(PoW, Proof-of-Work)이라는 합의 매커니즘을 제안하였는데, 각 블록의 헤더에 루트 해시, 이전 블록의 블록 해시, 그리고 난수라고 볼 수 있는 논스라는 값을 해싱하여 블록 해시를 구하는데, 이 과정에서 채굴자는 논스를 구하기 위해 수십억 번의 해시함수 연산을 시도해야 한다. 이처럼 컴퓨팅 자원을 소모하여 블록 생성 및 검증을 한다 하여 작업 증명이라 정의하고, 이러한 방식은 정직한 노드의 해쉬 레이트(i.e. 채굴을 위한 연산처리능력)가 악의적인 노드의 해쉬 레이트보다 높은 한 블록체인의 불변성은 보장된다. 블록체인에 기록되는 거래는 해당 블록체인에서 발행하는 암호화폐를 기반으로 하며, 이러한 암호화폐 거래 기록을 증명하는 노드들에게 보상을 주기 위해 암호화폐 거래 시에 수수료가 발생되게 된다. 즉,

암호화폐의 존재는 블록체인과 불가분의 존재이다. 자세한 블록체인의 자료 구조는 그림 2.1을 참고하면 된다.

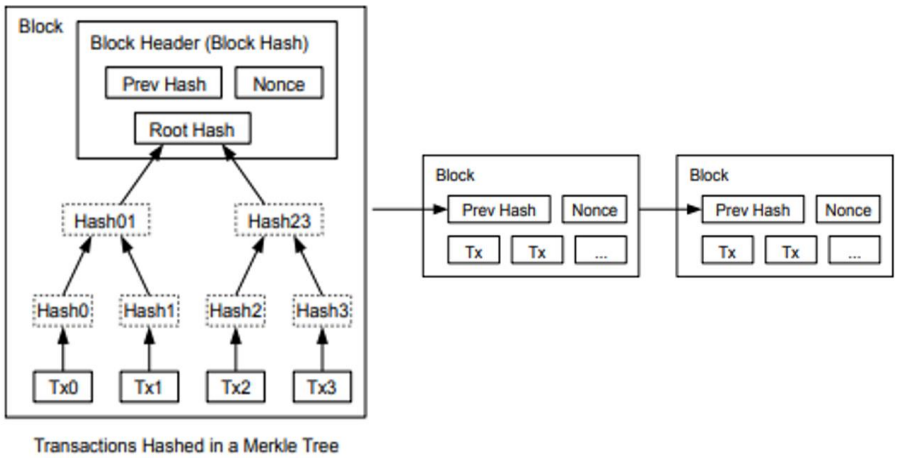

그림 2.1 Nakamoto[2] 연구에서 제안한 블록체인 자료 구조

이러한 합의는 King and Nadal [4]이 제안한 노드의 보유 암호화폐 수에 따라 블록 생성 권한이 비례하는 지분 증명(PoS, Proof-of-Stake)와 Larimer[5]가 제시한 노드에 위임된 암호화폐 수량에 따라 블록 생성 권한이 비례하는 위임지분 증명(DPoS, Delegated-Proof-of-Stake)으로도 이루어지기도 하나, 기반하는 블록체인의 자료 구조 역시 비슷한 방식으로 불변성을 확보한다.

2.2 이더리움 가상 머신과 스마트 컨트랙트

Buterin[6]은 최초의 암호화폐인 비트코인이 단순히 거래 증명만을 하는 것에서 더 발전된 형태의 암호화폐인 이더리움을 개발하는데, 이더리움의 가상 머신인 EVM(Ethereum Virtual Machine)에서는 비트코인의 UTXO(Unspent Transaction Output) 기반의 트랜잭션 관리 방식이 아닌, 기존의 은행 거래와 같이 계정 기반의 원장 관리 방식을

도입하고, 연산 코드 및 저장 데이터 등과 같은 자원에 대한 수수료(가스라고 통칭한다) 구조를 채택하였다. 이에 따라 블록체인 상의 암호화폐 전송 및 계정 자산의 변경에 특정한 instruction으로 지시할 수 있는 튜링완전성을 갖춘 코드를 지원한다. 이러한 코드를 Solidity, Vyper, Yul이라는 언어로 작성할 수 있으며, 자체 컴파일러로 컴파일한 바이트 코드는 EVM에 의해 실행 가능한 스마트 컨트랙트가 된다. 컨트랙트는 블록체인의 서버이자 계정으로, 암호화폐를 일반 사용자처럼 보유할 수 있다. 이더리움 트랜잭션의 상태 변화 로직은 그림 2.2.에서 확인할 수 있다.

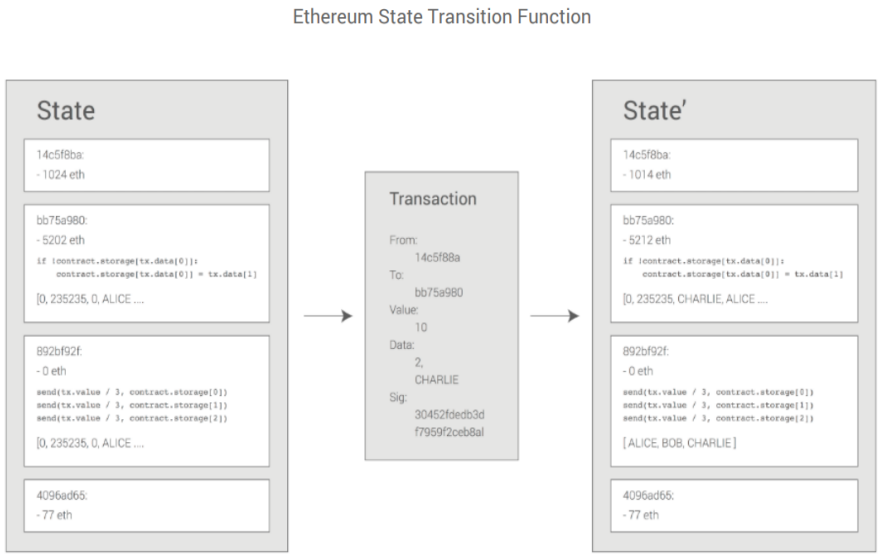

그림 2.2 Buterin[6] 연구에서 사용한 이더리움 트랜잭션의 도식

2.3 Tokenization을 통한 자산 유동화

기존에 존재하는 블록체인 위에 발행되는 통화를 주로 토큰이라고 부르는데, 대부분의 블록체인 플랫폼은 이러한 토큰을 발행하는 데에 코드의 표준(e.g. 이더리움의 ERC20)을 정해 호환을 정한다. 해당

토큰을 발행하는 것은 단순히 해당 블록체인의 데이터베이스에 단순한 데이터를 기록하고 저장하는 것에 불과하고, 이러한 토큰이 가치를 갖기 위해서는 이미 법정화폐로도 일정 수준의 가치가 인정된 암호화폐를 받아서 토큰을 판매하는 방식으로 가치를 pegging하였다. 이러한 방법으로 주로 ICO(Initial Coin Offering)가 사용되었지만, STO(Security Token Offering)의 ICO와의 차별점을 고려했을 때 머신러닝 모델을 유동화하는 방식으로서는 STO가 보다 적합하다고 판단하였다.

Lambert et al[7]은 증권형 토큰(Security Token)에 대해 블록체인에 기록된 투자 상품에 대한 디지털적 권리 혹은 표상이라고 설명하며, 해당 토큰은 마치 기존의 증권처럼 기반이 되는 자산에 대한 지분과 의결권을 가지기 때문에 기존의 ICO와 달리 지분으로서의 성격이 강하고 확실한 인센티브 구조를 가지고 있다고 한다. 머신러닝 모델 역시 토큰화를 할 경우 기존의 ICO를 통해 발행된 암호화폐들이 블록체인 플랫폼 자체를 제공하는 등의 유틸리티를 제공하는 것과 달리 블록체인 외부에 실물로 존재하는 자산을 유동화하는 방식의 토큰화이기 때문에, 각 토큰이 해당 자산에 대한 지분을 보유하는 방식의 STO가 적합한 유동화 방식일 것이다. 머신러닝 모델에 대한 지분은 “해당 머신러닝 모델을 특정 데이터 크기 혹은 연산량만큼 점유할 권리”로 치환될 수 있다. ICO를 통한 유틸리티 토큰과 STO를 통한 증권형 토큰의 차이는 표 2.1을 참고하면 된다.

표 2.1 Lambert et al[7] 연구의 ICO와 STO 비교

	Security token	Utility token
What	A security token is a digital representation of an investment product	A utility token gives consumptive rights to access a product or service
Why	The security token is bought by an investor with the expectation of a profit	The utility token is mainly issued to spend it in a developed, community-based ecosystem. It has similar features to a voucher
Who	The issuer usually is a for-profit entity, such as a private limited company. That company's objective is to create value for investors by selling products and services to customers	The original issuers were foundations, not for profit entities, of which tokenholders became community members. The product or service aims to create value for members
How	The issuance is usually launched through an STO and recorded on a distributed ledger commonly using blockchain technology	The issuance is usually launched through an ICO and recorded on a distributed ledger commonly using blockchain technology
When	Depending on the type of security, tokens can be issued when a start-up company can articulate information required for a securities prospectus. The issuance can also be conducted later, i.e., by established companies	Tokens are usually issued early, when the core team has conceptualized their community project and described it in a whitepaper
Whose	Security tokens fall under the purview of securities laws that is implemented by the respective financial regulator. It is very common to limit access to high-risk securities to accredited investors	Utility tokens are, in most jurisdictions, not considered as securities and, therefore, not governed by securities laws. This removes regulatory obstacles to access them for retail buyers. Since they are more akin to product vouchers, utility tokens are subject to consumer protection and tax laws

2.4 AMM을 통한 자동화 거래 시장

AMM(Automated Market Maker)는 Hanson[8]의 logarithmic market scoring rule(LMSR)에서 시작되었다. Hanson[8]은 해당 논문에서 자산을 예치하는 유동성 공급자를 두어 특정 비율로 자산 간의 가격이 자동적으로 시장에 의해 형성될 수 있는 아이디어를 제안하였고, 이를 기반으로 Zhang et al[9]이 constant product market maker를 제안하고 이를 직접 컨트랙트 코드상에 구현함으로써 탈중앙화 거래소 등에 널리 활용되고 있다. 구체적인 알고리즘을 살펴보면, α라는 암호화폐를 β로, 혹은 역으로 거래할 수 있는 시장을 형성하는 방식으로, 해당 시장에 각 암호화폐 예치액(Reserve) $R\alpha > 0$, $R\beta > 0$ 이라고 했을 때, 해당 reserve의 곱(product)이 특정 상수(constant)에 고정되도록 함으로써 자동화적으로 가격을 형성하는 것이다. 이는 식 2.1로 정의할 수 있다.

$$R\alpha * R\beta = k \qquad \text{(식2.1)}$$

Rα: 암호화폐 α의 잔고 혹은 예치 개수
Rβ: 암호화폐 β의 잔고 혹은 예치 개수
k: 고정 상수

즉, α와 β가 트레이딩 페어로 묶이게 되고, β를 Δβ > 0만큼 판매하여 Δα > 0만큼의 α를 구매할 경우 하기의 식 2.2.을 만족하여야 한다.

$$(R\alpha - \Delta\alpha) * (R\beta + \Delta\beta) = k \qquad \text{(식2.2)}$$

Δα: 구매하게 되는 α의 개수
Δβ: 판매하는 β의 개수

위와 같이 작동하여 Δα를 자동적으로 구할 수 있어 일반적인 거래에서 bidding을 하여 중개자가 매칭해주는 방식이 필요없이 자동적으로 거래를 성사시킬 수 있고, 또한 reserve에 자산 α와 β의 비율이 거래에 따라 계속 변동되어 시장 수요와 공급이 가격 형성에 반영되게 된다. 이러한 CPMM 전략은 그림 2.3과 같이 표현될 수 있다.

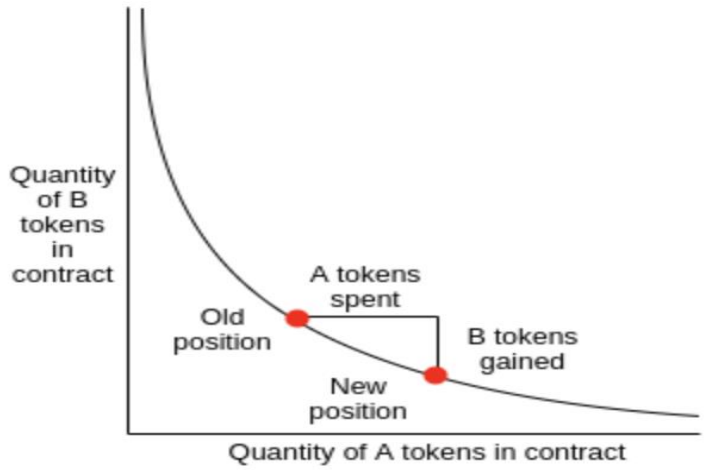

그림 2.3 Buterin[10]이 도식화한 CPMM 알고리즘

이러한 CPMM 전략은 reserve에 예치되는 유동성이 늘어날수록 price impact 및 slippage가 작아지며, 다양한 암호화폐의 시장가격을 형성하고 거래할 수 있는 마켓을 만드는 데에 큰 기여를 하고 있다.

2.5 데이터 오라클을 통한 온-오프체인 연결

Breidenbach et al[11]은 스마트 컨트랙트를 통해 블록체인 상에 존재하지 않는 외부 데이터를 요청하기 위한 데이터 오라클인 체인링크를 제안하였다. 합의 증명을 통해 데이터의 무결성이 입증되는 블록체인 상의 데이터와 달리, 외부의 데이터를 가져오고 검증하는 데에 대한 과정이 블록체인 자체에는 내재되어 있지 않다. 따라서, 그림 2.4에서 제안된 것과 같이 블록체인과 외부 데이터를 보유한 웹 서버 사이에 탈중앙화 오라클 네트워크를 두어 데이터를 검증하고 외부 데이터를 호출할 수 있도록 한다.

기존의 전통적 오라클이 그림 2.5와 같이 각각의 검증자가 각자 데이터를 검증하는 것과 달리, 체인링크의 오라클은 2.6과 같이 검증자 노드들이 네트워크를 구성하여 마치 독립적인 블록체인과 같이 검증의 과정을 거친다.

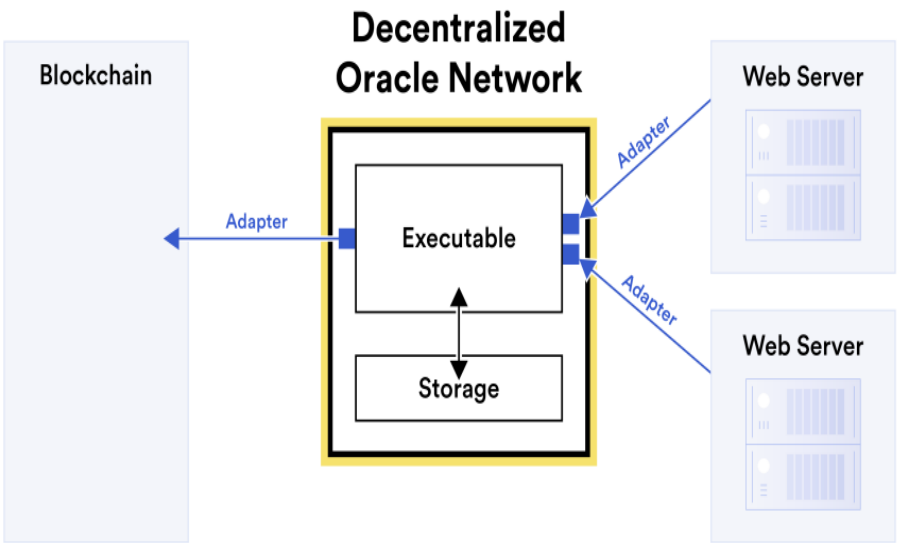

그림 2.4 Breidenbach et al[11]에서 사용된 아키텍처

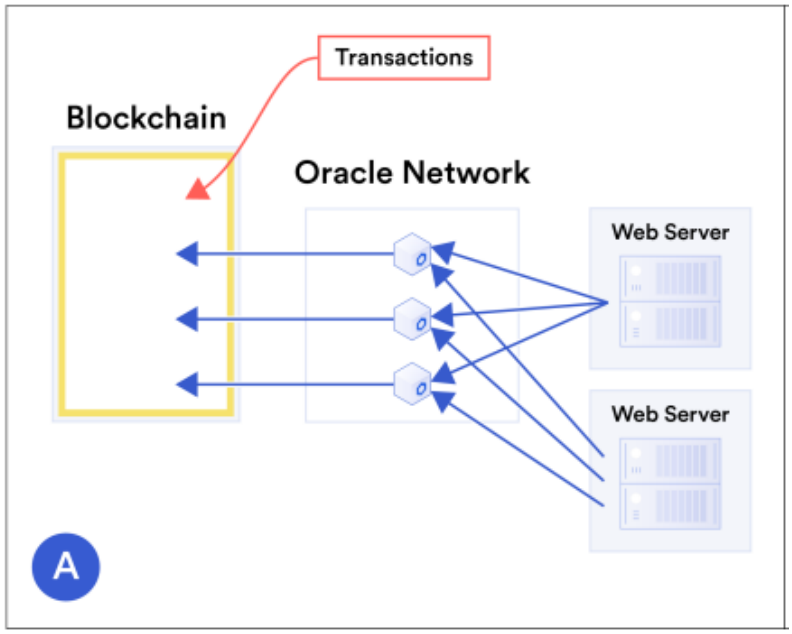

그림 2.5 Breidenbach et al[11]에서 사용된 전통적 오라클 도식

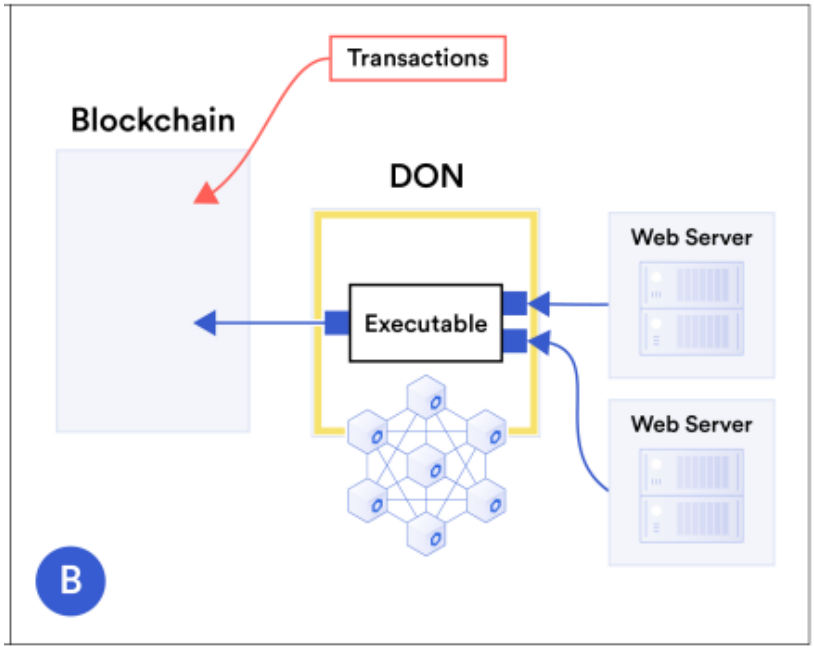

그림 2.6 Breidenbach et al[11]이 제안한 오라클 도식

제 3 장 제안하는 방법

3.1. CPMM을 통한 AI 모델 유동화 및 거래

배포할 블록체인 네트워크로는 이더리움을 선택하였다. 이더리움 블록체인 상의 스마트 컨트랙트는 솔리디티로 작성되며, 바이트 코드로 컴파일 되어 이더리움 노드에 로딩되어 기존의 웹 서버에서 작동하는 프로그램과 유사하다. 그러나 스마트 컨트랙트는 이더리움이라는 암호화폐를 주고받을 수 있으며, 하나의 계정으로서도 작동하는 점에서 차이가 있다.

CPMM은 다음과 같이 활용될 수 있다. CPMM 로직을 구현한 스마트 컨트랙트 계정에 유동화하고자 하는 AI 모델 A에 대한 지분을 갖는 A 토큰을 이더리움 토큰 표준인 ERC20으로 발행한다. 이에 대한 STO는 해당 컨트랙트에 이더리움을 전송함으로써 진행될 수 있다. 전체 예치된 이더리움 대비 전송한 이더리움의 비율이 A 토큰에 대한 지분이 된다. 이후 AI 모델을 통해 데이터를 분석하고자 하는 클라이언트는 A 토큰을 이더리움으로 구매하여 권리를 얻을 수 있다. 이와 같이 AI 모델 토큰을 이더리움으로 거래 시의 CPMM 알고리즘은 그림 3.1과 같이 pseudocode로 표현된다.

Algorithm 1 Swap Model Token or Ethereum in CPMM

CPMM Contract $\leftarrow C$

Contract Call Wallet $\leftarrow W$

Balance of Model Token $\leftarrow M_{Balance}(C)$

Balance of Ethereum $\leftarrow E_{Balance}(C)$

Constant $K \leftarrow M_{Balance}(C) \times E_{Balance}(C)$

if $Type(AmountIn) = M$ **then**

$AmountOut \leftarrow E_{Balance}(C) - K/(M_{Balance}(C) + AmountIn)$

transfer $AmountOut$ to W

update

$E_{Balance}(C)' \leftarrow E_{Balance}(C) - AmountOut$

$M_{Balance}(C)' \leftarrow M_{Balance}(C) + AmountIn$

else if $Type(AmountIn) = E$ **then**

$AmountOut \leftarrow M_{Balance}(C) - K/(E_{Balance}(C) + AmountIn)$

transfer $AmountOut$ to W

update

$E_{Balance}(C)' \leftarrow E_{Balance}(C) + AmountIn$

$M_{Balance}(C)' \leftarrow M_{Balance}(C) - AmountOut$

그림 3.1. AI 모델 토큰 거래 시 CPMM pseudocode

3.2. 데이터 오라클을 통한 학습 요청

본 연구의 경우, 기존의 머신러닝과 관련한 연구들에서 성능의 최적화 등에 초점을 둔 것과 달리 머신러닝 모델을 상품으로서 유동화하고 블록체인 상에서 거래할 수 있는 아키텍처 및 코드를 구현함에 있다. 따라서, 학습의 프로세스를 Mock 서버로 대체하여 해당 서버에 요청 시 데이터 크기와 학습 정확도를 JSON 형식으로 반환해주도록 하였다. 이는 외부의 데이터이기 때문에 체인링크의 스마트 컨트랙트를 상속하여 해당 오라클 함수를 사용하여 데이터 무결성을 입증하였다.

3.3. 전체 아키텍처 및 프로세스

CPMM, 오라클, 학습 서버를 통한 최종적인 프로세스는 다음과 같다. 먼저 머신러닝 연구원 A는 자신의 모델 A를 이용할 수 있는 권한(i.e. 클라이언트 데이터를 전달해줄 경우 모델 A로 분석을 해줄 수 있는)을 가진 암호화폐 A를 발행한다. 해당 토큰 A를 이미 법정화폐로서도 가치가 연동 되어있는 암호화폐(e.g. 이더리움, 이오스) B를 자금으로 받는 STO를 진행하여 한 컨트랙트 reserve에 예치한다. 해당 컨트랙트는 토큰 A와 암호화폐를 예치받는 계정이자 CPMM 알고리즘을 코드로 구현한 블록체인 서버이다. STO 참여자는 해당 모델 권한에 대한 지분을 갖는다. 이 때 A에 대한 초기 가격 및 공급량 산정은 모델 A에 따라 매우 달라질 수 있고 임의적이다. 본 논문에서는 오라클의 파라미터로 데이터의 크기를 받아 지불해야하는 A의 개수와 비례하도록 이용 권한도 데이터의 크기에 따라 다른 개수의 A를 지불하도록 할 수 있다.

이후 클라이언트는 토큰을 지불함으로써 이용할 수 있는 AI 모델에 대한 설명 등을 보고, 위 2번의 CPMM으로 시장가격이 형성된 A 토큰을 B 암호화폐로 구매한다. 이러한 거래가 빈번해질 경우 A토큰의 가격은 자동으로 상승하게 된다.

A 모델 토큰 매수가 완료되면 학습 프로세스가 시작된다. 따라서, 클라이언트는 연구원에게 분석하고자 하는 데이터를 전송하고, 컨트랙트에 학습 요청을 하면, 컨트랙트는 오라클을 활용하여 학습 서버에 요청을 한다. 이후 학습 결과를 요약해서 보여주는 정확도를 블록체인 상에 기록하고, 요청된 데이터의 크기를 파라미터로 받아와 클라이언트가 연구원에게 A 토큰을 지불하도록 한다. 상기의 프로세스는 그림 3.2와 같은 아키텍처를 가진다.

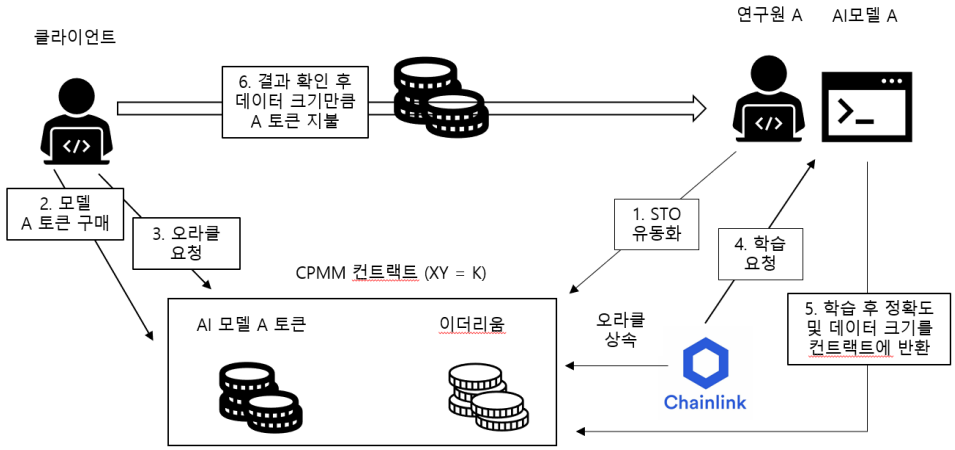

그림 3.2. 제안하는 아키텍처 및 프로세스

제 4 장 실험 및 분석

4.1. 실험 환경

실험 환경은 크게 스마트 컨트랙트가 작동하는 블록체인 서버와 유동화된 모델 토큰을 구매하여 학습하고자 하는 클라이언트, 그리고 학습 서버로 구성되는데, 먼저 클라이언트 측 환경에서는 이더리움 블록체인에 로드된 스마트 컨트랙트에 RPC(Remote Procedure Call)를 JSON 규격으로 통신하는 자바스크립트 라이브러리인 ethers.js를 활용하여 프로그램을 실행하는 것으로 설계하였다. 사용자 경험을 위해 브라우저를 통해 GUI로 실행할 수 있지만 본 논문의 목적과는 무관하므로 고려하지 않았다. 같은 이유로 학습 서버 역시 Postman이라는 개발 도구를 사용해 토큰 지불 개수에 사용될 데이터 크기와 블록체인 상의 기록을 위한 정확도를 반환해주는 Mock 서버 형식으로 구축하였다.

블록체인 서버의 경우에는 일반적인 웹 어플리케이션 서버와 유사하게 작동하는데, 스마트 컨트랙트를 작성하여 가상머신에 로드할 경우 해당 컨트랙트는 서버에서 실행하는 프로그램이자 하나의 계정이 된다. 이더리움 블록체인 위에서 실험을 진행하기 위해 스마트 컨트랙트는 Solidity로 작성하였으며, 배포할 블록체인 네트워크로는 이더리움 Rinkeby 테스트 네트워크를 선택하였다. 2021년 런던 하드포크로 업그레이드된 가장 최신의 이더리움 가상 머신 환경에서 0.8.14 버전의 컴파일러을 사용하였다. 스마트 컨트랙트 및 관련 트랜잭션 로그를 추적할 수 있는 블록탐색기로는 Etherscan을 사용하였다. 실제 실험에 사용된 세부적인 실험 조건은 표 4.1과 같다.

표 4.1 실험에 사용된 환경

스마트 컨트랙트	블록체인 플랫폼	Ethereum
	가상 머신	London Ethereum Virtual Machine
	작성 언어	Solidity
	컴파일러	0.8.14+ commit.80d49f37
	블록체인 네트워크	Rinkeby testnet
	블록 탐색기	Etherscan
클라이언트	CPU	Intel® Core™ i7-10700 @ 2.90HZ x 16
	OS	Ubuntu 20.04.3 LTS
	프로그램 작성 언어	JavaScript ES6
	통신 프로토콜	JSON-RPC
	RPC 라이브러리	Ethers.js
학습 서버	Mock 서버 구축 도구	Postman

4.2. 스마트 컨트랙트 구현 및 배포

본 연구의 아키텍처에서 핵심이 되는 것은 해당 AI 모델이 STO를 통해 토큰화되고 토큰화된 AI 모델이 특정한 중개자 없이 CPMM 알고리즘에 기반하여 자동으로 거래가 될 수 있는 시장을 만드는 것이다. 이는 EVM에 의해 해석될 수 있는 스마트컨트랙트를 제공함으로써 가능해지며, 이러한 컨트랙트에 적절한 양식을 갖추어 RPC를 통해 이더리움 블록체인 상의 데이터베이스를 조회, 변경하는 것이 가능해진다.

이를 구현하기 위한 이더리움 상의 스마트 컨트랙트를 ModelTokenFactory라는 Solidity로 구현하였으며, ModelToken에 대한 symbol, name, supply를 생성자의 파라미터로 지정함에 따라 해당 모델을 유동화하고자 하는 연구원이 적절한 명칭과 공급량을 정할 수 있다. 또한 이더리움 상에서 스마트 컨트랙트는 사용자의 계좌와

마찬가지로 주소를 가지고 암호화폐를 해당 컨트랙트 자체가 소유할 수 있도록 하였다. 해당 컨트랙트는 배포될 시에 유동화된 ModelToken을 하기와 같이 이더리움으로 구매 혹은 판매할 수 있는 함수를 포함하고 있으므로, 적절한 수량의 이더리움을 해당 컨트랙트에 전송하는 것만으로 유동화가 되고 CPMM에 기반한 자동화 거래가 가능해진다. 변수 및 생성자를 선언한 솔리디티 코드는 그림 4.1, CPMM 거래 방식을 구현한 솔리디티 코드는 그림 4.2와 같다.

이러한 ModelTokenFactory 솔리디티 코드를 Remix 개발 툴을 사용하여 컴파일하였으며 이더리움 Rinkeby 테스트 네트워크에 배포하였다. 배포 시 그림 4.4와 같이 머신러닝을 유동화한 모델 토큰의 이름, 심볼, 발행량을 파라미터로 주입하여 배포하였고 해당 배포 트랜잭션의 로그를 확인할 수 있다. 이더리움 네트워크 상의 지갑, 스마트 컨트랙트 등의 정보를 확인할 수 있는 익스플로러인 Etherscan을 통해 확인한 결과 그림 4.5과 같이 배포한 ModelTokenFactory 컨트랙트 계정에 SOTA_MODEL이라는 이름의 ERC20 토큰이 예치되어 있으며, 해당 컨트랙트 주소에 일정 수량의 이더리움을 전송할 경우 해당 수량의 비가 바로 pegging된 초기 가격으로 책정되게 된다.

또한 체인링크라는 오라클 서비스를 상속하여 학습 서버에 정확도와 데이터 크기를 요청하고 받아오는 로직을 구현하였다. 컨트랙트의 requestAccuracyAndData 함수를 통해 학습 서버에 요청을 한 후, 해당 학습 서버에서 결과 값을 json으로 반환해주면 체인링크에서 fulfill 함수를 호출하여 컨트랙트 상의 변수를 초기화하는 방식이다. 이 때 체인링크가 자체적으로 발행한 LINK 토큰을 지불한다. 이를 구현한 프로그램의 솔리디티 코드는 그림 4.3과 같다.

```
/// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import '@chainlink/contracts/src/v0.8/ChainlinkClient.sol';
import '@chainlink/contracts/src/v0.8/ConfirmedOwner.sol';

contract ModelTokenFactory is ChainlinkClient, ConfirmedOwner,
    Ownable {
    // Using Chainlink Oracle
    using Chainlink for Chainlink.Request;

    uint256 public accuracy;
    uint256 public dataSize;
    bytes32 private jobId;
    uint256 private fee;

    // AI model token
    IERC20 public modelToken;
    // AI model developer
    address public developer;
    // Synchronized lock
    uint private unlocked = 1;
    modifier lock() {
        require(unlocked == 1, 'LOCKED');
        unlocked = 0;
        _;
        unlocked = 1;
    }
    // AI model developer deploy contract with parameter name,
        symbol, supply
    // Initialize Chainlink Oracle
    constructor(string memory name, string memory symbol, uint
        totalSupply) ConfirmedOwner(msg.sender) {
        modelToken = new ERC20(name, symbol, totalSupply * 10 **
            18);
        developer = msg.sender;

        setChainlinkToken(0
            x01BE23585060835E02B77ef475b0Cc51aA1e0709);
        setChainlinkOracle(0
            xf3FBB7f3391F62C8fe53f89B41dFC8159EE9653f);
        jobId = '53f9755920cd451a8fe46f5087468395';
        fee = (1 * LINK_DIVISIBILITY) / 10; // 0,1 * 10**18 (Varies
             by network and job);

        modelToken.mint(address(this), totalSupply);
    }
```

그림 4.1. ModelTokenFactory 변수 및 생성자 선언 코드

```
// Contract can receive ethereum which will peg to model token
    to monetize
receive() external payable {}
// Buy AI model token with ethereum
function swapEthToModel() external payable lock {
    uint k = address(this).balance * modelToken.balanceOf(
        address(this));
    uint ethAmount = msg.value;
    require(ethAmount > 0, "Amount must be >0");
    uint modelTokenAmount = modelToken.balanceOf(address(this))
         - k/(address(this).balance + ethAmount);
    modelToken.transfer(msg.sender, modelTokenAmount);
}
// Buy ethereum with AI model token
function swapModelToEth(uint modelTokenAmount) external lock {
    uint k = address(this).balance * modelToken.balanceOf(
        address(this));
    require(modelTokenAmount > 0, "Amount must be >0");
    uint ethAmount = address(this).balance - k/(modelToken.
        balanceOf(address(this)) + modelTokenAmount);
    modelToken.transferFrom(msg.sender, address(this),
        modelTokenAmount);
    payable(msg.sender).transfer(ethAmount);
}
```

그림 4.2. ModelTokenFactory CPMM 코드

```
    // Request accuracy result to machine learning server by using
        Chainlink
    function requestAccuracyAndData() public {
        Chainlink.Request memory req = buildChainlinkRequest(jobId,
            address(this), this.fulfillMultipleParameters.selector
            );

        // Set the URL to perform the GET request on
        req.add('urlAccuracy', 'URL');
        req.add('urlDataSize', 'URL');
        // Set path for json
        req.add('pathAccuracy', 'Accuracy');
        req.add('DataSize', 'DataSize');

        // Sends the request
        sendChainlinkRequest(req, fee);
    }

    // Receive the response called by Chainlink
    function fulfill(bytes32 _requestId, uint256 _accuracy, uint256
         _dataSize) public recordChainlinkFulfillment(_requestId) {
        accuracy = _accuracy;
        dataSize = _dataSize;
    }

    // Allow withdraw of Link tokens from the contract
    function withdrawLink() public onlyOwner {
        LinkTokenInterface link = LinkTokenInterface(
            chainlinkTokenAddress());
        require(link.transfer(msg.sender, link.balanceOf(address(
            this))), 'Unable to transfer');
    }

}
```

그림 4.3. ModelTokenFactory 오라클 코드

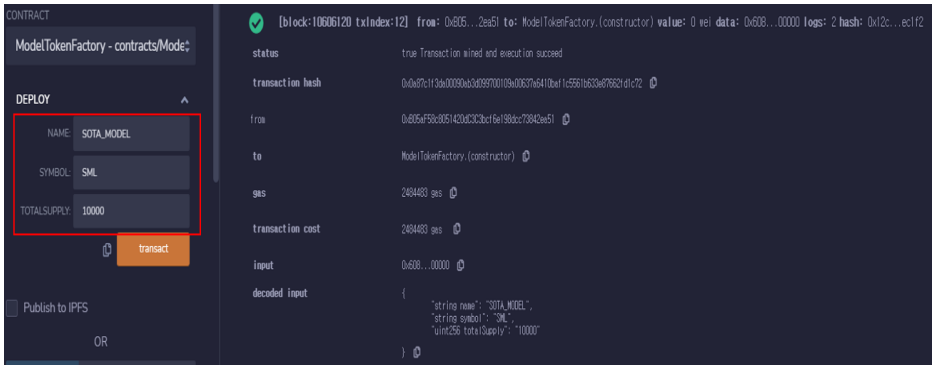

그림 4.4. ModelTokenFactory 배포 로그

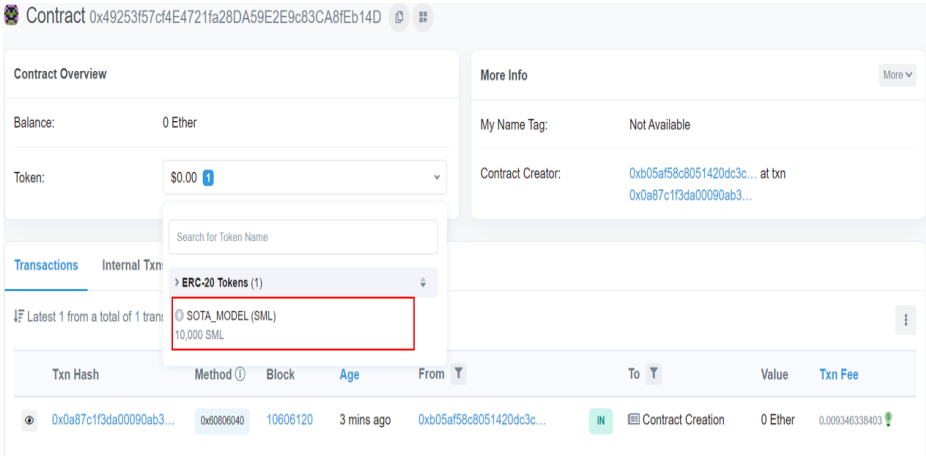

그림 4.5. ModelTokenFactory 컨트랙트 계정 정보

4.3. 자동화 프로그램 구현

이더리움의 노드는 컨트랙트 바이트 코드를 실행시킬 수 있는 JSON-RPC를 제공하여 기존의 브라우저 혹은 웹서버와 통신할 수 있도록 한다. Ethers.js 라이브러리를 사용하여 EVM에 로드된 컨트랙트 바이트 코드에 RPC를 요청할 수 있고, 해당 라이브러리를 활용하여 자바스크립트 기반 자동화 프로그램 구현하였다. 컨트랙트 코드인 ModelTokenFactory를 컴파일하고 배포하는 프로세스를 자동화하여 AI 모델을 토큰으로 유동화하는 프로그램 코드는 그림 4.6과 같다. 함수 deployAiContract에 유동화할 모델 토큰의 이름, 심볼, 발행량을 파라미터를 넣어 실행한다.

```
// private key of wallet which calls this contract
const privateKey = "privateKey";
// Set rpc node
const provider = new ethers.providers.JsonRpcProvider("
    ethereumRpcUrl");
// Set client wallet
const signer = new ethers.Wallet(privateKey, provider);

// Method to deploy ModelTokenFactory which monetize AI model
function deployAiContract(tokenName, tokenSymbol, tokenSupply) {
    // Set aiContractFactory to deploy
    const aiContractFactory = new ethers.ContractFactory(
        aiContractAbi, ByteCode, signer);
    // Set parameter for aiContract
    const aiContract = await aiContractFactory.deploy(
    tokenName, tokenSymbol, tokenSupply);
    // Set address for aiContract
    const aiContractAddress = aiContract.address;
    // Deploy and log
    console.log(await aiContract.deployTransaction.wait());
}
```

그림 4.6. 컨트랙트 배포 자동화 자바스크립트 코드

또한 아키텍처에서 제안한 토큰 구매, 학습, 지불 프로그램 코드는 4.7과 같다. 구체적인 프로세스는 클라이언트의 요청 시, 먼저 이더리움으로 유동화된 AI 모델 토큰을 CPMM 알고리즘을 기반으로 구매한다. 이후 컨트랙트의 오라클 함수를 통해 학습 서버에 요청하여 클라이언트 데이터를 학습한 후, 정확도와 데이터 크기 값을 받는다. 마지막으로 머신러닝을 통해 분석이 요청된 데이터의 크기만큼 구매된 모델 토큰을 해당 모델을 개발한 연구원에게 전송한다.

```
// Method to buy Model Token and pay for running distributed
    learning
function transactionForAi(etherAmount) {
    // Ai contract
    const aiContract = new ethers.Contract(aiContractAddress,
        aiContractAbi, signer);
    // Ai model token contract
    const erc20TokenContract = new ethers.Contract(tokenAddress,
        ERC20Abi, signer);
    // swap Eth for Model Token
    await aiContract.swapEthToModel({value: ethers.utils.parseEther
        (etherAmount)});

    // Machine learning process begins
    let dataSize;
    // Call global server to get data size and learning result
    aiContract.requestAccuracyAndData();

    while (aiContract.getDataSize() === null) {
        datasize = aiContract.getDataSize();
    }
    // Machine learning process ends

    // transfer model token
    const fromAmount = ethers.utils.parseEther(dataSize);
    await erc20TokenContract
    .transfer(aiContract.getDeveloper(), fromAmount);

}
```

그림 4.7. 모델 구매, 학습, 지불 자동화 자바스크립트 코드

4.4. 결과 분석

위와 같이 AI 모델을 스마트 컨트랙트를 통해 블록체인 상의 토큰으로 유동화하고, 해당 모델을 활용하기 위해 토큰을 구매하고 사용하는 일련의 과정을 프로그램화하였다. 이러한 연구의 가장 큰 목적은 첫 번째로, 해당 모델의 사용도가 높아질수록 모델 토큰의 가격이 상승하는 것이 CPMM이라는 알고리즘에 의해 자동으로 책정될 수 있는 환경을 만들어 AI 모델이 시장 경제의 평가를 받을 수 있도록 한 것이고, 두 번째로는 블록체인 상에 해당 모델의 가격 추이에 더해 정확도와 같은 학습 결과를 기록하여 투명성과 신뢰성을 제고함에 있다.

설계한 CPMM 알고리즘을 기반으로 한 스마트 컨트랙트 코드의 검증을 위해 다음과 같은 시나리오를 활용하여 모델 토큰의 가격의 변화 정도를 살펴보았다.

1) 유동화된 모델 토큰 SOTA_MODEL과 이더리움이 스마트

컨트랙트 계정에 각 100개씩 예치되어 1:1의 가격비로 설정

2) AI 모델을 활용하고자 하는 클라이언트 100명이 1 이더리움으로 모델 토큰 SOTA_MODEL 구매, 지불, 학습 프로그램 순차 실행

3) 시나리오 동안 그 외의 클라이언트가 모델 토큰 SOTA_MODEL 구매 혹은 판매하는 경우는 없는 경우로 전제

위와 같이 실행한 결과 CPMM 컨트랙트가 정상적으로 작동하여 모델 토큰의 가격이 선형적으로 상승하는 것을 그림 4.8과 같이 확인할 수 있다.

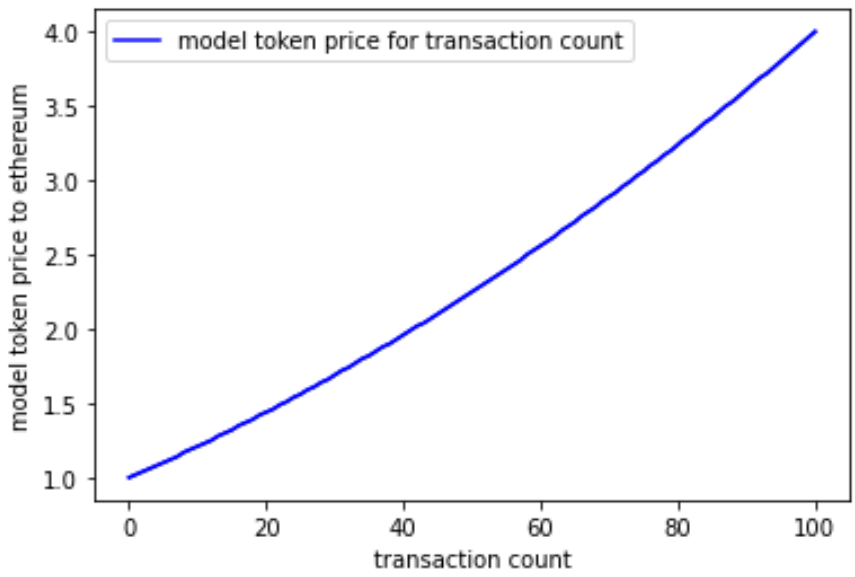

그림 4.8. 트랜잭션 카운트에 따른 모델 토큰의 가격 추이

제 5 장 결론

5.1. 결론

본 연구를 통해 무형화된 자산이지만 매우 가치가 높은 자산인 머신러닝 모델이 어떻게 유동화될 수 있고, 이에 따라 기존의 블랙박스식의 거래에서 벗어나 블록체인의 분산원장 기술을 통해 퍼블릭하게 기록되고, 또 CPMM 전략에 따라 자동으로 해당 모델의 가격이 산정될 수 있는 방법에 대해 제시하였다. 이러한 방법을 통해 모델이 암호화폐로 유동화되게 하며, 거래에 따라 가격이 상승 혹은 하락할 수 있는 시장이 형성됨에 따라 시장 가격의 흐름으로 해당 모델의 효과성 및 효율성을 정량적으로 평가할 수 있게 됨에 의의가 있다. 또한 모델의 분석 결과를 블록체인에 투명하게 기록하는 것은 신뢰성을 제고할 것이다. 신뢰성을 제고하기 위한 오라클 서비스를 구현한 것 역시 의의가 있을 것이다.

5.2. 향후 연구에 대한 논의

본 연구에서는 AI 모델을 유동화하고 클라이언트와 모델을 개발한 측 간의 거래를 블록체인 상의 스마트 컨트랙트로 자동화하였지만, CPMM 이외의 다양한 자동화 거래 알고리즘을 고려하지 않았고, 블록체인 밖에서 진행되는 학습에 대한 설계에 대한 고려가 부족한 측면이 있다. 또한 새롭게 제안하는 아키텍처에 대한 연구이고, 프로그램을 구현하였으나 어떠한 평가 지표로 실험을 진행하고 평가할지에 대한 점이 아직 모호한 한계가 있다.

추후 연구를 통해 상기의 한계점을 극복하고 아키텍처를 보다 구체적으로 실현하기 위해서는 1) 각 모델 특성에 맞는 초기 유동성 형성 전략 2) 슬리피지 최소화를 위한 유동성 공급 전략 3) 학습 프로그램 설계 구현 4) 적절한 평가 지표 선정 5) UI 구현을 통한 실용성 제고 등에 대한 연구가 추가적으로 이루어질 필요가 있을 것으로

사료된다.

참고 문헌

[1] Pino, P., Parra, D., Messina, P., Besa, C. Uribe, S. (2020), Inspecting state of the art performance and NLP metrics in image-based medical report generation, In Proceedings of Workshop: LXAI Research NeurIPS.

[2] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review, 21260.

[3] Eastlake, D. and Jones, P. (2001). US secure hash algorithm 1 (SHA1). IETF Requsest for Comments 3174.

[4] King, S. and Nadal, S. (2012, August). Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. self-published paper. vol. 19. no. 1.

[5] Larimer, D. (2014). Delegated proof-of-stake (dpos). Bitshare whitepaper. vol. 81. p. 85.

[6] Buterin, V. (2014). A next-generation smart contract and decentralized application platform. Ethereum whitepaper. vol. 3. no. 37.

[7] Lambert, T., Liebau, D., Roosenboom, P. (2021). Security token offerings. Small Bus Econ.

[8] Hanson, R. (2003). Combinatorial information market design, Information Systems Frontiers, vol. 5, no. 1, pp. 107–119.

[9] Zhang, Y., Chen, X., Park, D. (2018). Formal specification of constant product (xy=k) market maker model and implementation.

[10] Buterin, V. (2014). The x*y=k market maker model. Retrieved from https://ethresear.ch/t/improving-front-running-resistance-of-x-y-k-market-makers.

[11] Breidenbach, L., Cachin, C., Chan, B., Coventry, A. Ellis, S., Juels, A., Koushanfar, F., Miller, A., Magauran, B., Moroz, D., Nazarov, S., Topliceanu, A., Tramer, F., Zhang, F. (2021). Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks, Chainlink whitepaper. vol. 1.0

Abstract

Monetization of Machine Learning Model and Architecture for Automated Market

Hyunhum Cho
Department of Industrial Engineering
The Graduate School
Seoul National University

A lot of recent research and paper insists its performance accomplish SOTA(State-of-the-Art). However, most of the result only perform SOTA with optimized experiment data and environment. As a result, machine learning model cannot be successfully applied to industry. This paper suggests architecture which is based on the concept of machine learning, smart contract and STO(Security Token Offering). By tokenizing machine learning model, the value of model can be intuitively and objectively estimated. Besides, demand for machine learning model can be automated by smart contract code, while protect client data privacy via distributed learning. Solidity and Javascript program is implemented to prove the proposed architecture.

Keywords : Machine Learning, Automated Market, Blockchain, STO
Student Number : 2020-24316

감사의 글

서울대학교 산업공학과의 모든 분들에게 감사드립니다.